



UM EECS 270 F22

Introduction to Logic Design

8. Combinational Building Blocks

Combinational Blocks



- Goal is to create a “toolbox” of devices that are frequently required in logic design
 - Originally, each device would be on its own IC
 - Today, circuit can be made so small that many circuits can fit on a single chip
 - VLSI can *stantiate* adders, comparators, etc., as many times as needed within a larger circuit
 - We would like to identify other frequently used combinational circuits and find small circuits for them in advance

Equality Comparator



- Want a signal, EQ, that is 1 iff two 4-bit numbers A and B, are equal. Under what logic conditions are A and B equal?
 - Equal if $a_3 = b_3$ AND $a_2 = b_2$ AND $a_1 = b_1$ AND $a_0 = b_0$
- Which gate performs an equality comparison?

A	B	$A \oplus B$
0	0	1
0	1	0
1	0	0
1	1	1

$$EQ = (a_3 \oplus b_3) \cdot (a_2 \oplus b_2) \cdot (a_1 \oplus b_1) \cdot (a_0 \oplus b_0)$$



How many gates are required to implement an n -bit “parallel” comparator?

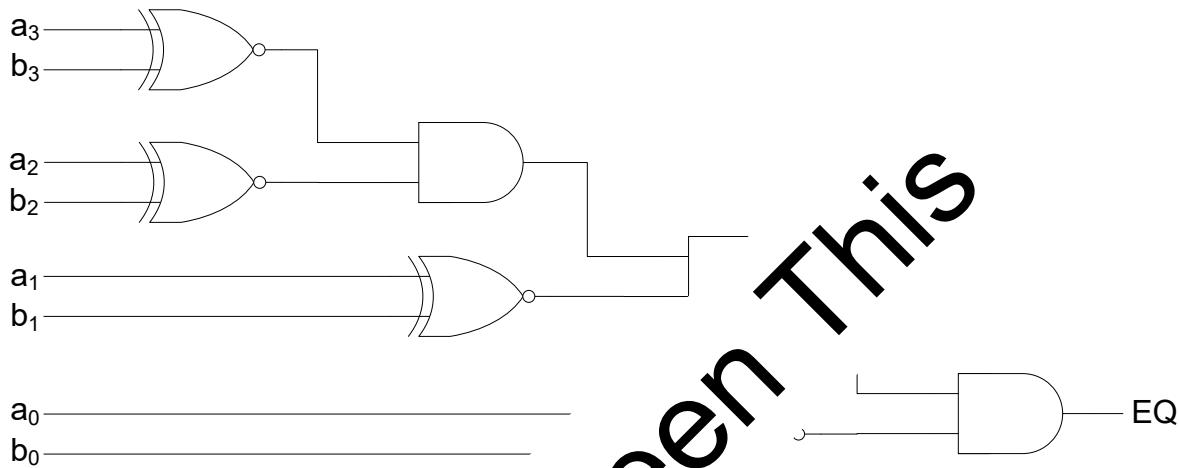
Assume $n = 2^k$

$$\#gates = 2^k + 2^{k-1} + 2^{k-1} + \dots + 2^0$$

$$= 2^{k+1} - 1$$

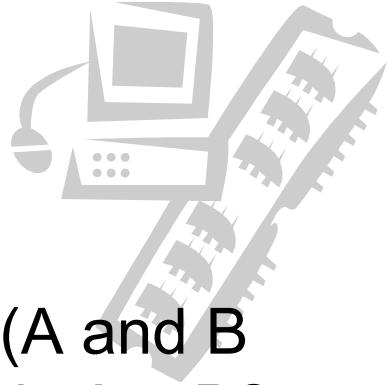
$$= 2n - 1$$

A “Series” Comparator Implementation



How many gates are required to implement an n -bit comparator?

- Comparing first two bits: 2 gates
 - Comparing each subsequent bit: 2 gates
 - After comparing the first 2 bits, there are $n-2$ bits left to compare for a total of $2*(n-2)$ gates
- Total number of gates required: $2(n-2) + 3 = 2n - 1$
- “Series” and “parallel” implementations require the same number of gates! Is there a reason to prefer one implementation over the other?



Magnitude Comparator

- Want to design a signal, $AgrB$, that is true if $A > B$ (A and B are 4-bit numbers). Under what logical conditions is $A > B$?
 - $A > B$ if $a_3 = 1$ and $b_3 = 0$
 - $A > B$ if $a_2 = 1$ and $b_2 = 0$ and $a_3 = b_3$
 - $A > B$ if $a_1 = 1$ and $b_1 = 0$ and $a_2 = b_2$ and $a_3 = b_3$
 - $A > B$ if $a_0 = 1$ and $b_0 = 0$ and $a_1 = b_1$ and $a_2 = b_2$ and $a_3 = b_3$

$AgrF$

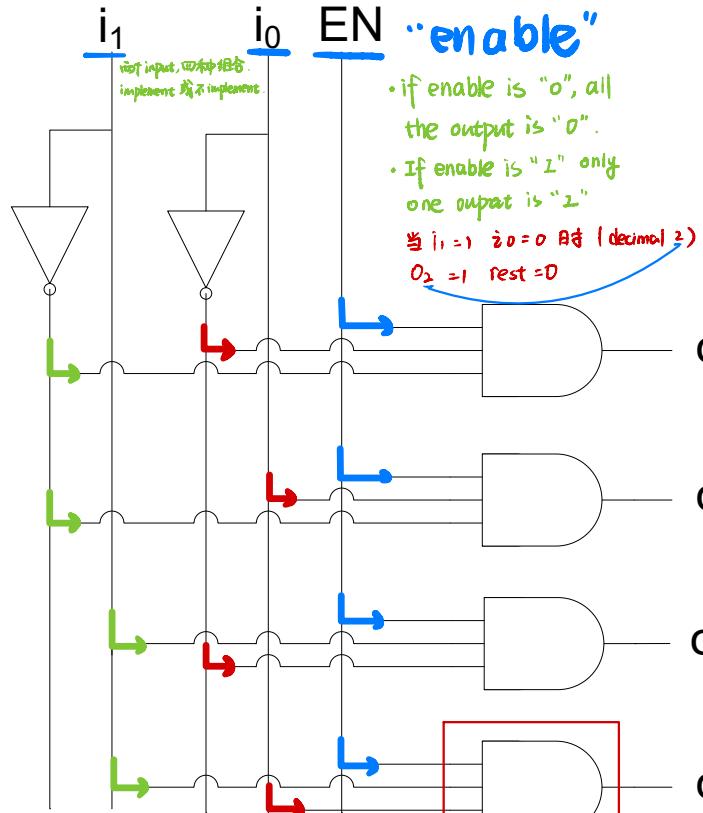
$$= (a_3 \odot b_3) \cdot a_2 \cdot \overline{b_2} + \\ (a_3 \odot b_3) \cdot (a_2 \odot b_2) \cdot a_1 \cdot \overline{b_1} + \\ (a_3 \odot b_3) \cdot (a_2 \odot b_2) \cdot (a_1 \odot b_1) \cdot a_0 \cdot \overline{b_0}$$

Decoder Circuit



two input

control input



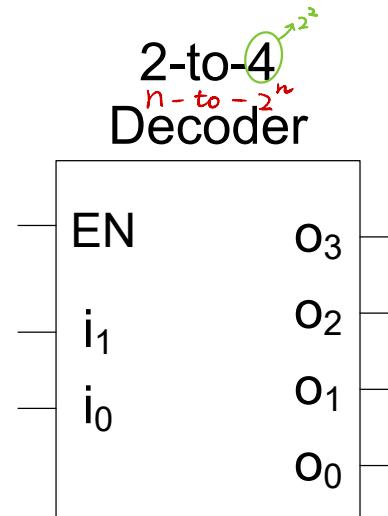
但随着输入的不断增加
“and” gate 也会不断变宽。
但是我们不可能有 32bit 的
“and” gate

4 output

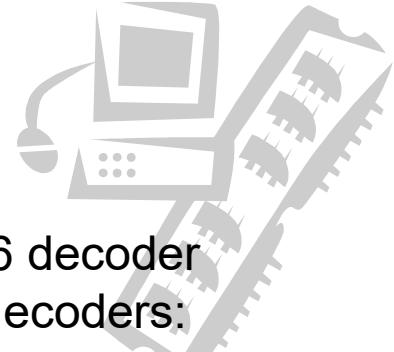
如果不看 EN 的话，这四个由 i_1, i_0 组成的表达式就是 minterm

→ 下一页：解决方法。

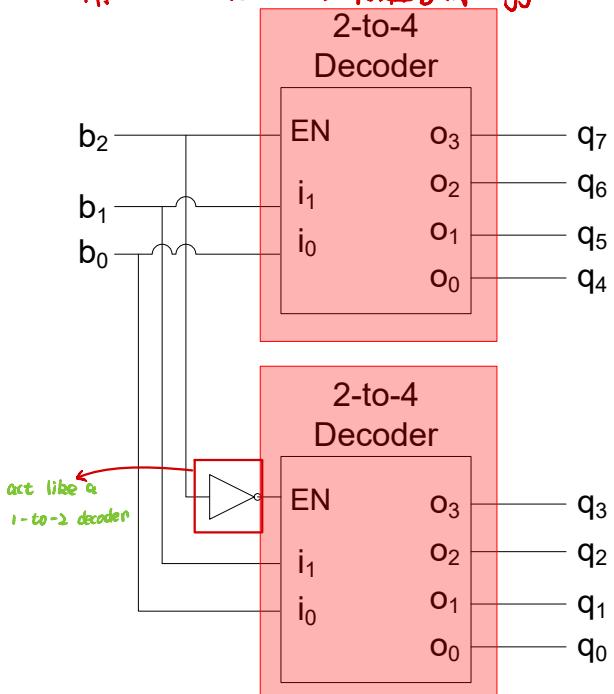
Symbol:



Cascading Decoders



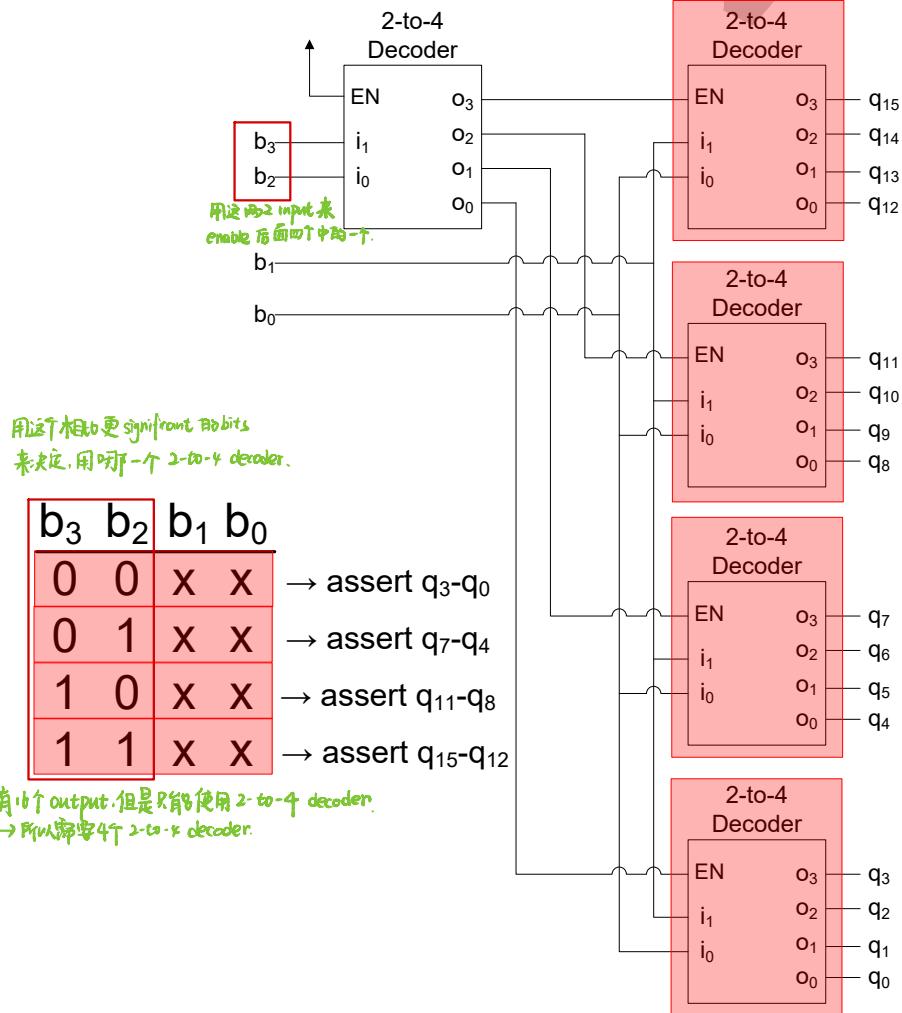
Creating a 3-to-8 decoder
from two 2-to-4 decoders:
用 smaller decoders 来组合成 bigger decoders



MSB 来决定是使
用第1个 2to4 encoder
还是第2个 2to4 encoder

b_2	b_1	b_0	意思是：使它变 成“1”，either 这两个中一个。
0	X	X	→ assert q_3-q_0 ($194 - 91 = 0$) 因为不enable.
1	X	X	→ assert q_7-q_4

Creating a 4-to-16 decoder
from five 2-to-4 decoders:

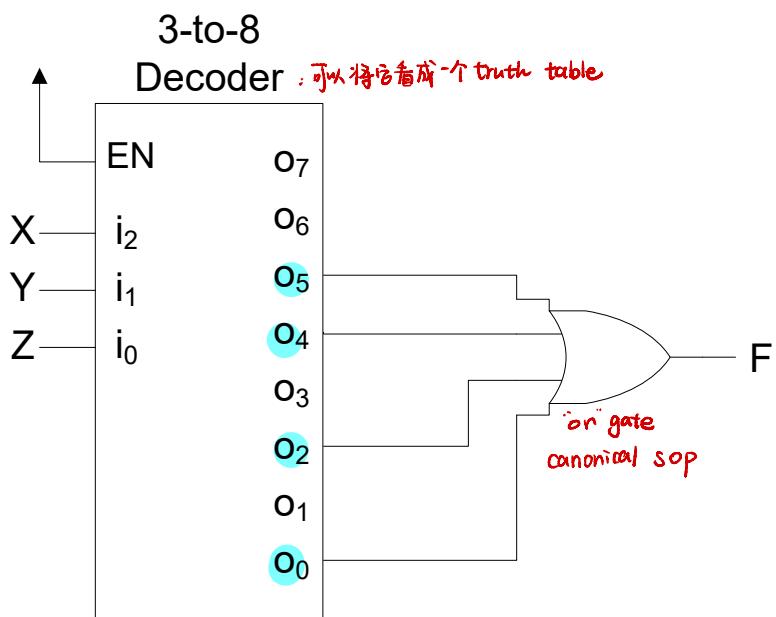




Implementing Functions with Decoders

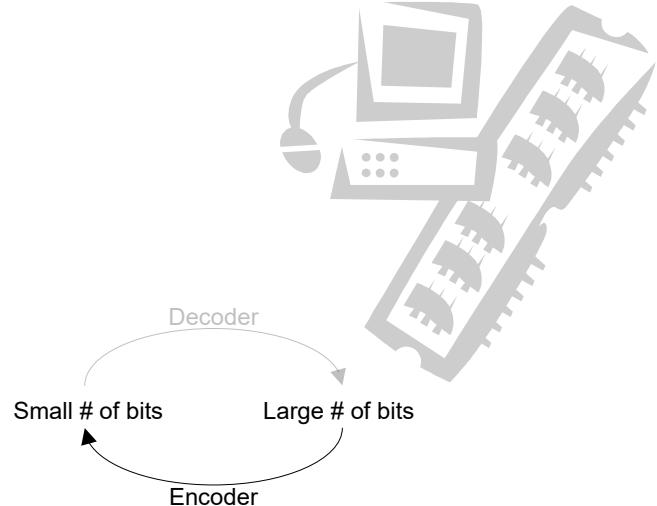
- Any n-variable function can be implemented with a n-to- 2^n decoder!
 - Recall that each decoder output is implemented as a minterm of the input variables
 - Recall that any function can be implemented as a sum of minterms
 - Therefore, any function can be implemented as a sum of decoder outputs!

$$F = \sum_{X,Y,Z} (0,2,4,5)$$



Encoders

- Opposite of decoders: an encoder takes as input a code with a large number of bits and outputs a corresponding code with a small number of bits
- Most common encoder maps a 2^n one-hot code to an n-bit binary number, where the binary number represents the input number that is asserted
- What if all input signals are deasserted? Requires additional output, usually called A (active) that is asserted if *at least* one input is asserted



4-to-2 encoder 但我们assume they are one hot

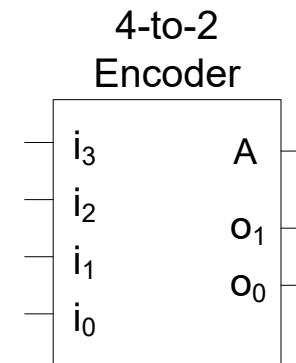
i_3	i_2	i_1	i_0	O_1	O_0	A
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	1	0	0	1	0	1
1	0	0	0	1	1	1
all others				d	d	d

当我们有4个input时,会有16种可能的combination.
但我们规定一个组合中最多只能有1个“1”出现。
(当然,没有1也是可以的)

但有一个问题是当input全为0时和当0时output都是00
该如何区分?
所以我们需要额外的output
• 当单个input为1时, Active为1
• 当所有input为0时, Active为0

$O_1 = i_3 + i_2$
 $O_0 = i_3 + i_1$
 $A = i_3 + i_2 + i_1 + i_0$

Encoder就是很多的“01”组成的



- What if we want to allow *multiple* inputs to be asserted simultaneously and output the asserted input number with the highest priority?
 - This task is performed by a **priority encoder**

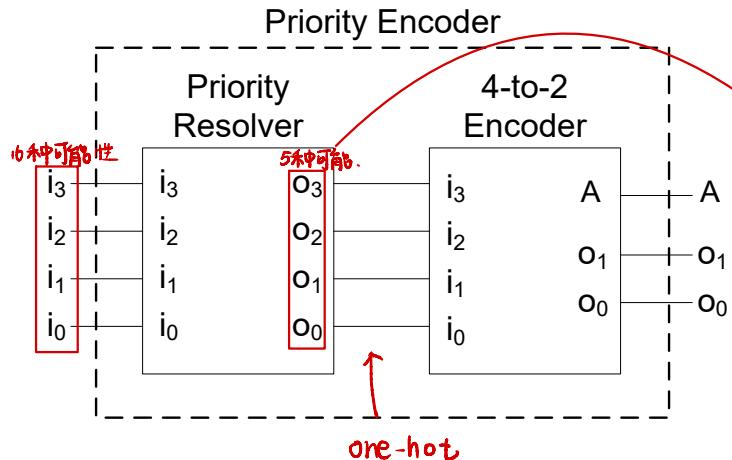
For priority $i_3 > i_2 > i_1 > i_0$

Highest priority
 $\xrightarrow{\text{x means either 0 or 1}}$

i_3	i_2	i_1	i_0	o_1	o_0	A
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

优先级
最高优先级 i_0, i_1, i_2, i_3 (Priority)

- An easy way to implement: use our existing “toolbox” – specifically, make use of the encoder that we’ve already designed
- Need to add a circuit that will resolve priority: assert output signal corresponding to asserted input with highest priority



Priority Resolver

i_3	i_2	i_1	i_0	o_3	o_2	o_1	o_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	X	0	0	1	0
0	1	X	X	0	1	0	0
1	X	X	X	1	0	0	0

$$o_3 = i_3$$

$$o_2 = i_2 \cdot \overline{i_3}$$

$$o_1 = i_1 \cdot \overline{i_2} \cdot \overline{i_3}$$

$$o_0 = i_0 \cdot \overline{i_1} \cdot \overline{i_2} \cdot \overline{i_3}$$

Multiplexer (Mux)

- A **mux** is a digital switch
- The output copies one of n data inputs, depending on the value of the *select inputs*
- Implementation:

$$q = s_1' s_0' d_0 + s_1' s_0 d_1 + s_1 s_0' d_2 + s_1 s_0 d_3$$

selectors are mutually exclusive
因为 minterm are mutually exclusive

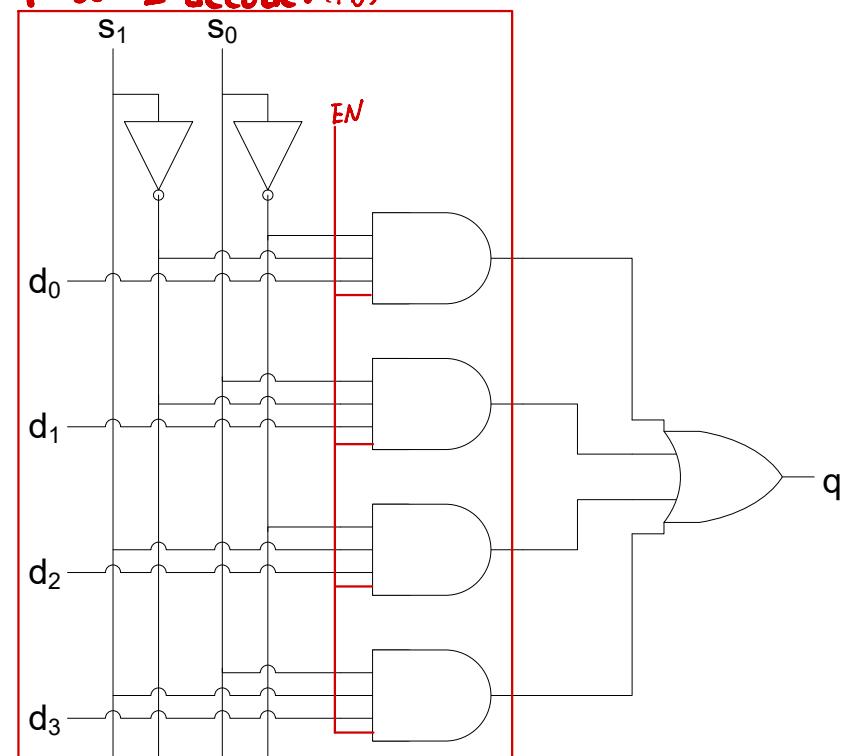
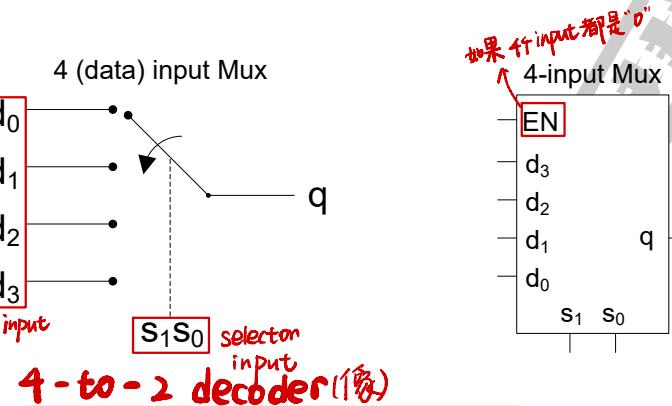
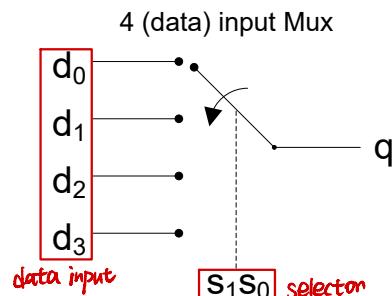
- 次只能选一个.

每次输入一个 $s_1 s_0$ 的组合.
对于四个 $s_1 s_0$ 的 minterm
来说, 只有 1 个是 1, 其余一定
是 0. 因此只需匹配每个

input d 和 $s_1 s_0$ 的 minterm
就能实现 4-to-1 的选择.

Product terms are mutually exclusive!

这四个 term, - 次只有一个最终主导结果



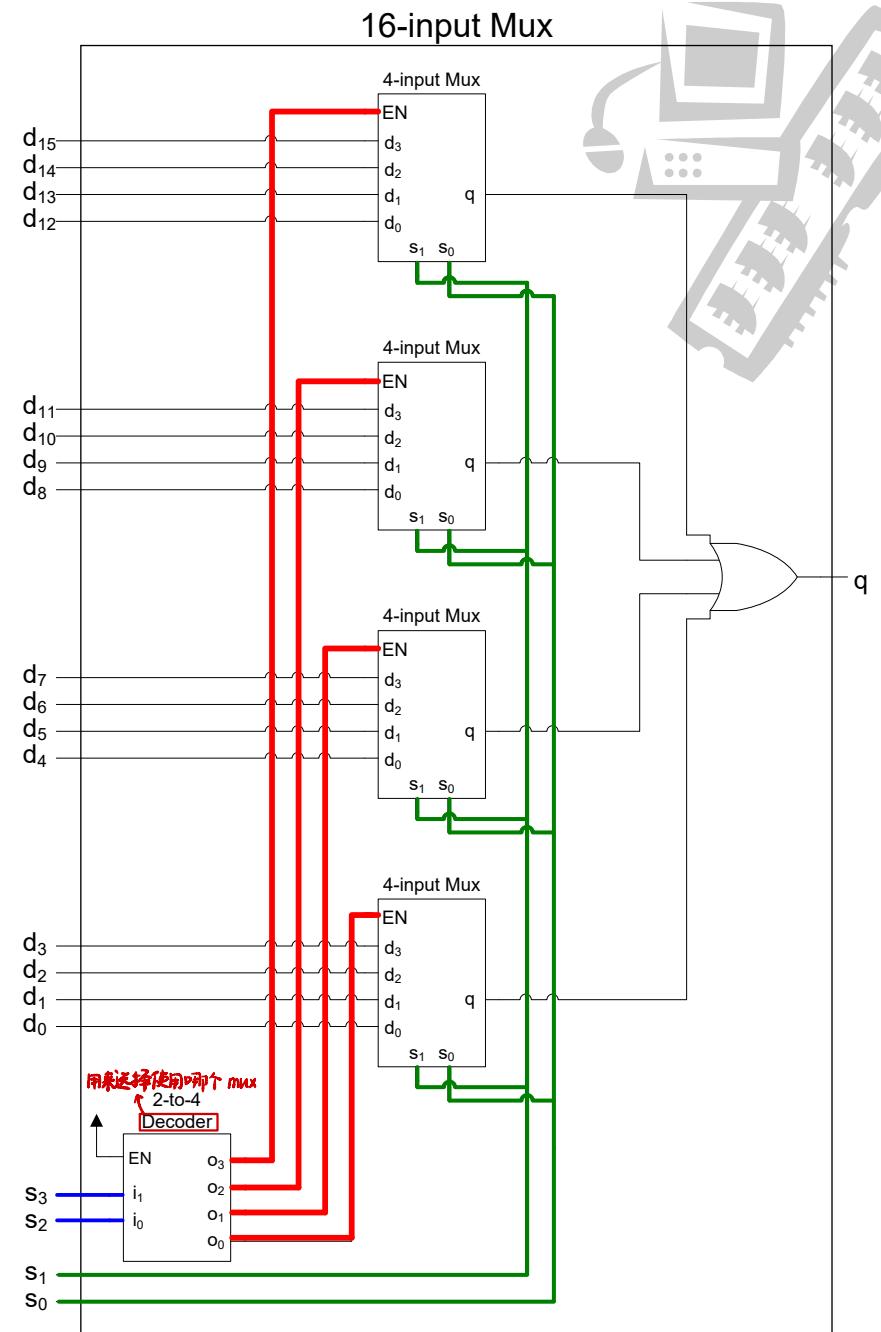
Building large multiplexers from smaller multiplexers (and other blocks)

...using four 4-input muxes and 1 2-to-4 decoder

S_3	S_2	S_1	S_0
0	0	X	X
0	1	X	X
1	0	X	X
1	1	X	X

用来选择
4个 mux 用哪个

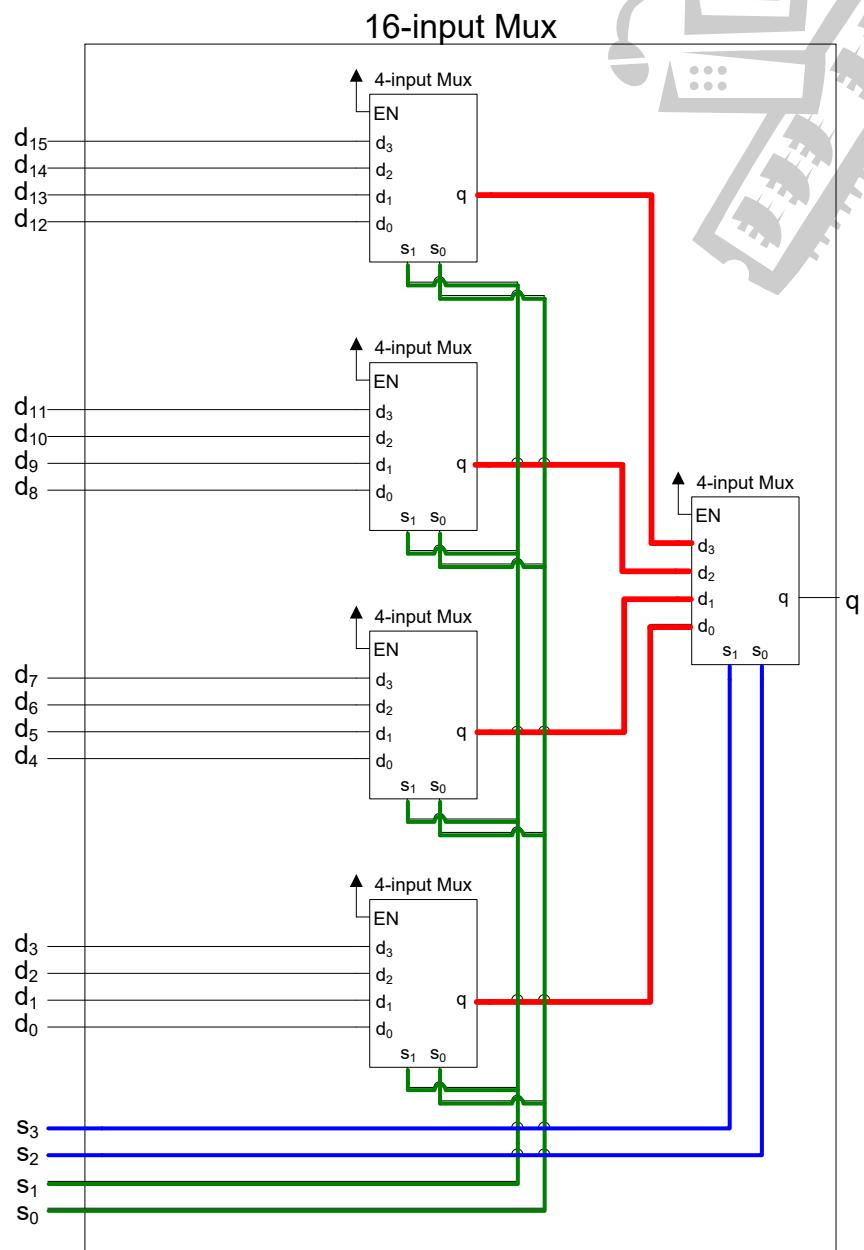
- select d_3-d_0
- select d_7-d_4
- select $d_{11}-d_8$
- select $d_{15}-d_{12}$



...using five 4-input muxes

S_3	S_2	S_1	S_0
0	0	X	X
0	1	X	X
1	0	X	X
1	1	X	X

- select d_3-d_0
- select d_7-d_4
- select $d_{11}-d_8$
- select $d_{15}-d_{12}$

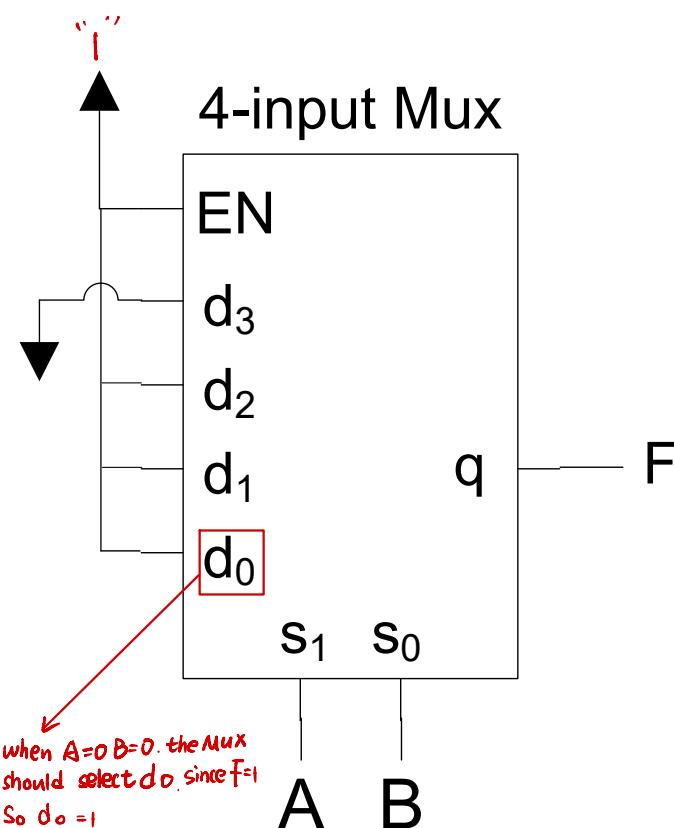




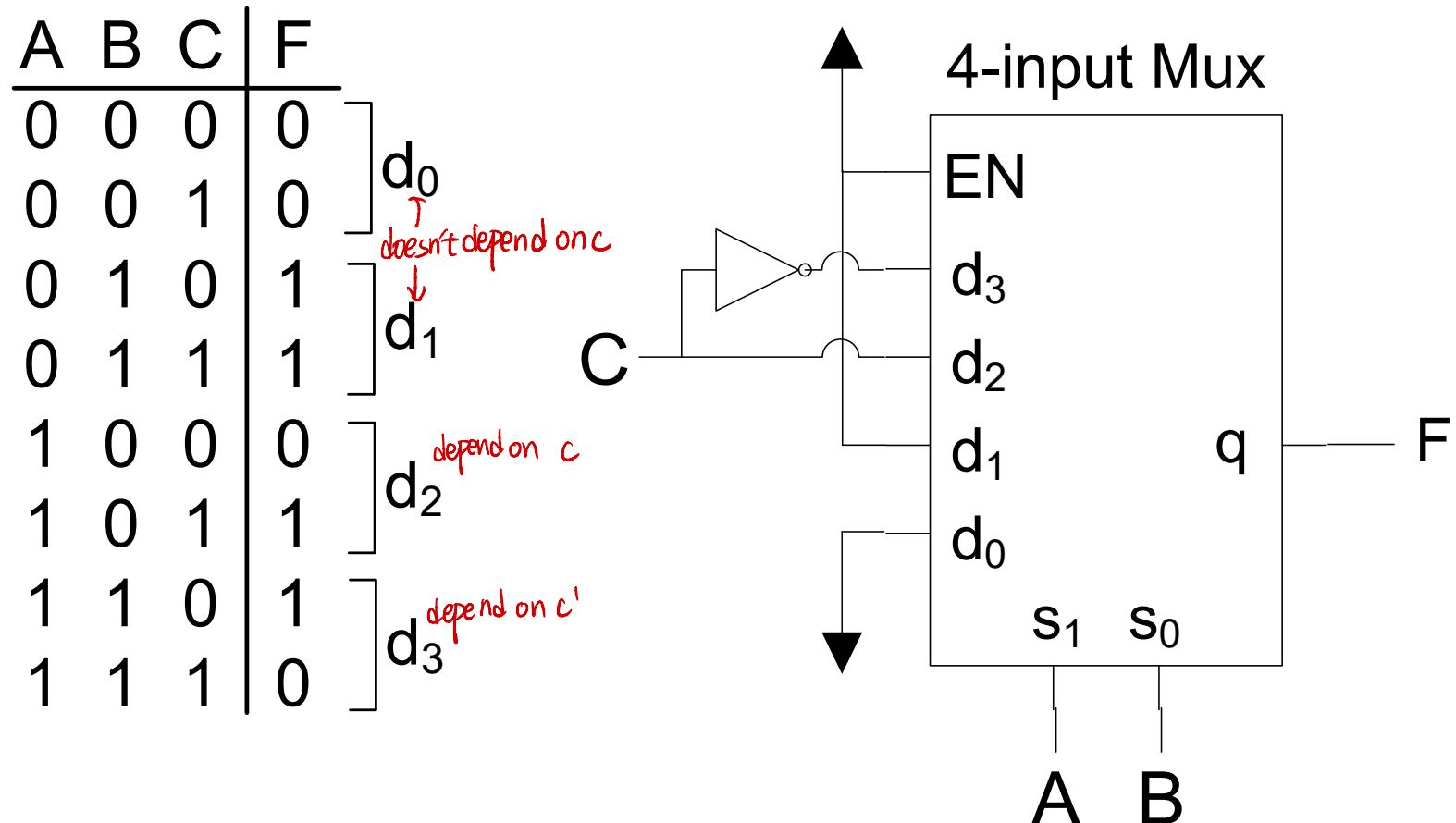
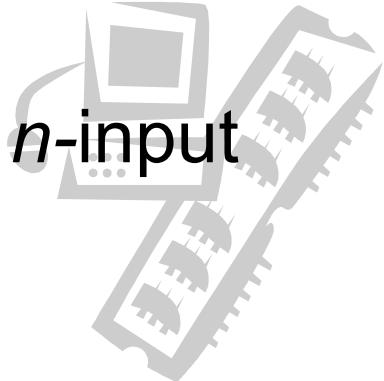
Implementing Functions With a Multiplexer

- We can implement an n -variable function using an n -select (2^n -input) mux

A	B		F
0	0		1
0	1		1
1	0		1
1	1		0



- We can do better: we can implement an n -input function with a 2^{n-1} -input mux!

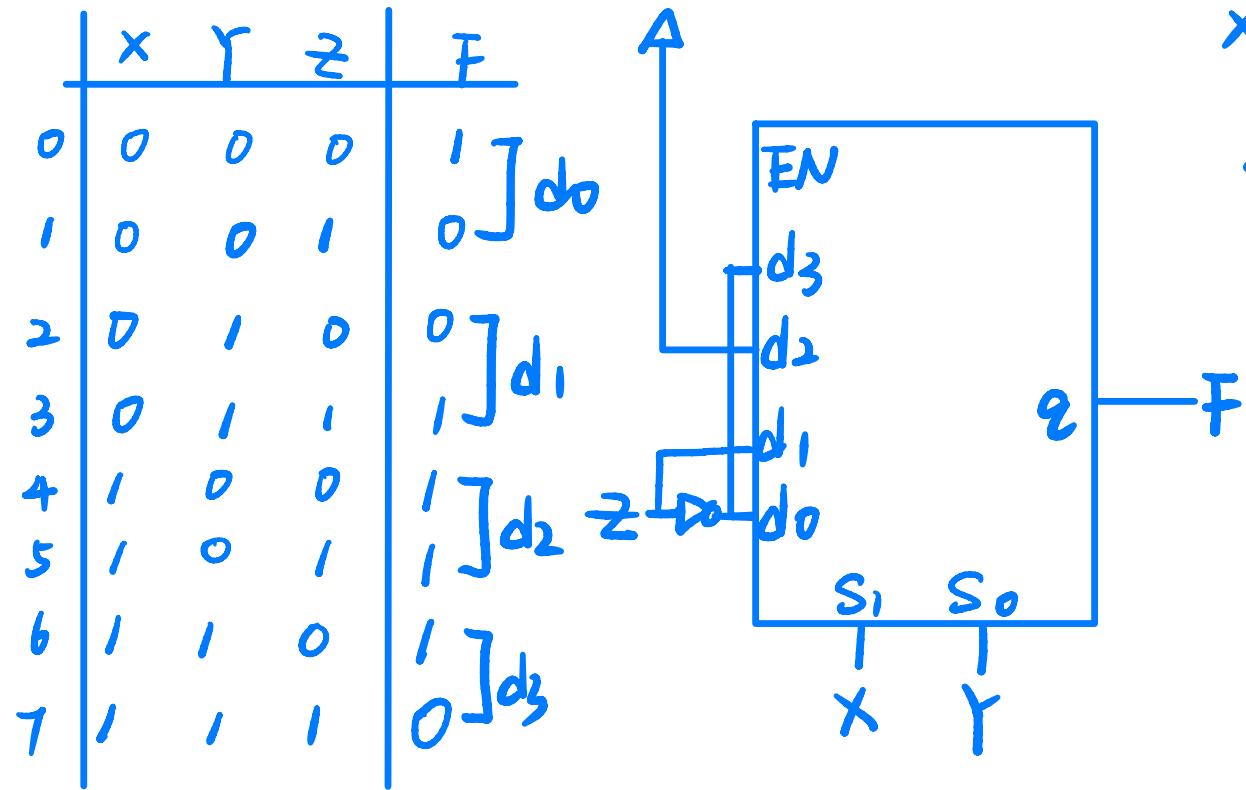


In Class Exercise



Implement the following 3-input function using a single 4-input mux:

$$F = \sum_{X,Y,Z} (0,3,4,5,6)$$



$$x'Y'(z') + x'Y(z) + \\ xy'(1) + xy.z'$$

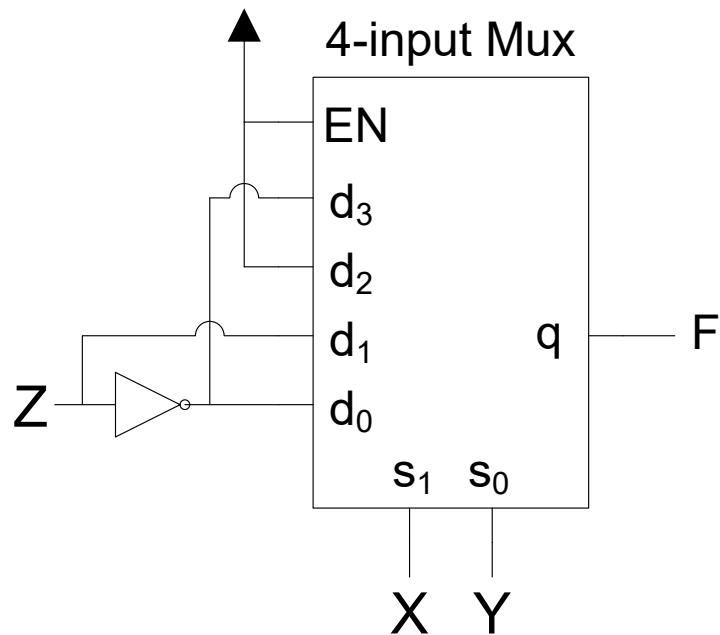
In Class Exercise



Implement the following 3-input function using a single 4-input mux:

$$F = \sum_{X,Y,Z} (0,3,4,5,6)$$

X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



Function Implementation with MUX is Application of Boole/Shannon Expansion Theorem



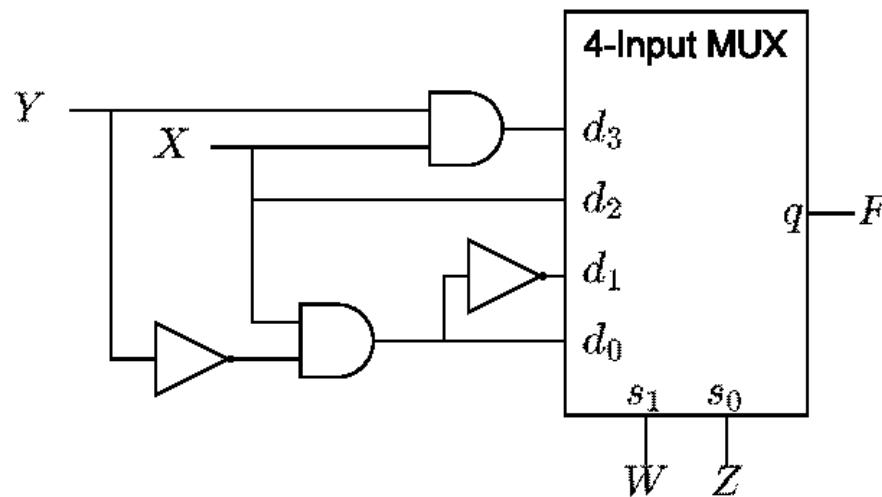
$$F(W, X, Y, Z) = W'X'Z + XY'Z' + XYZ + WXY$$

$$= W'Z'(\text{?}) + W'Z(\text{?}) + WZ'(\text{?}) + WZ(\text{?})$$

$$= \underline{W'Z'(XY')} + W'Z(X' + XY) + WZ'(XY' + XY) + WZ(XY)$$

当 $W=0$ $Z=0$ 将 $W=0$ 代入 function
看剩余的是什么。

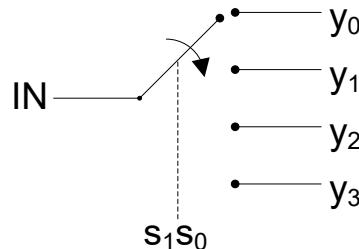
$$= W'Z'(XY') + W'Z(X' + Y) + WZ'(X) + WZ(XY)$$



Demultiplexer (demux)



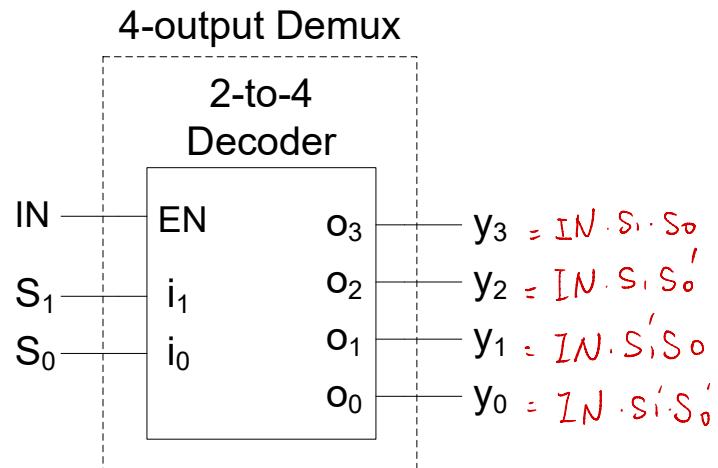
- A **demux** connects an input signal to one of several output signals, depending on the value of the select signals



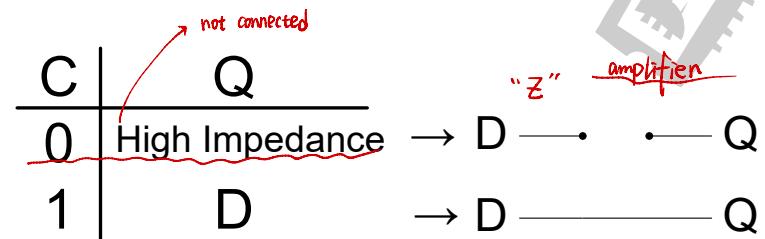
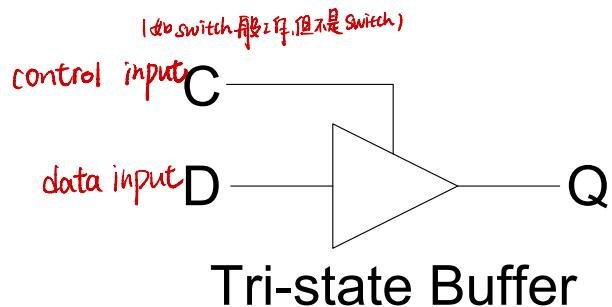
s_1	s_0	y_0	y_1	y_2	y_3
0	0	IN	0	0	0
0	1	0	IN	0	0
1	0	0	0	IN	0
1	1	0	0	0	IN

2-to-4 Decoder			
EN	i_1	i_0	O_0
0	x	x	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

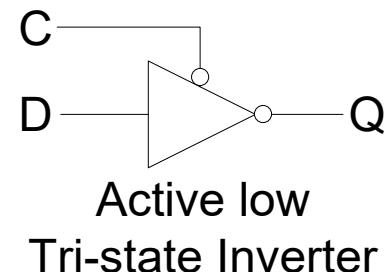
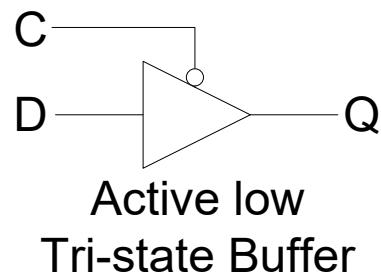
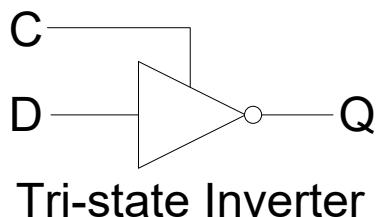
- How to implement?
 - TT should look very familiar...
 - Use a decoder!



3 Tri-State Gates



Other tri-state devices:



C	Q
0	High Impedance
1	\bar{D}

C	Q
0	D
1	High Impedance

C	Q
0	\bar{D}
1	High Impedance

- Is it ok to connect the outputs of tri-state devices?

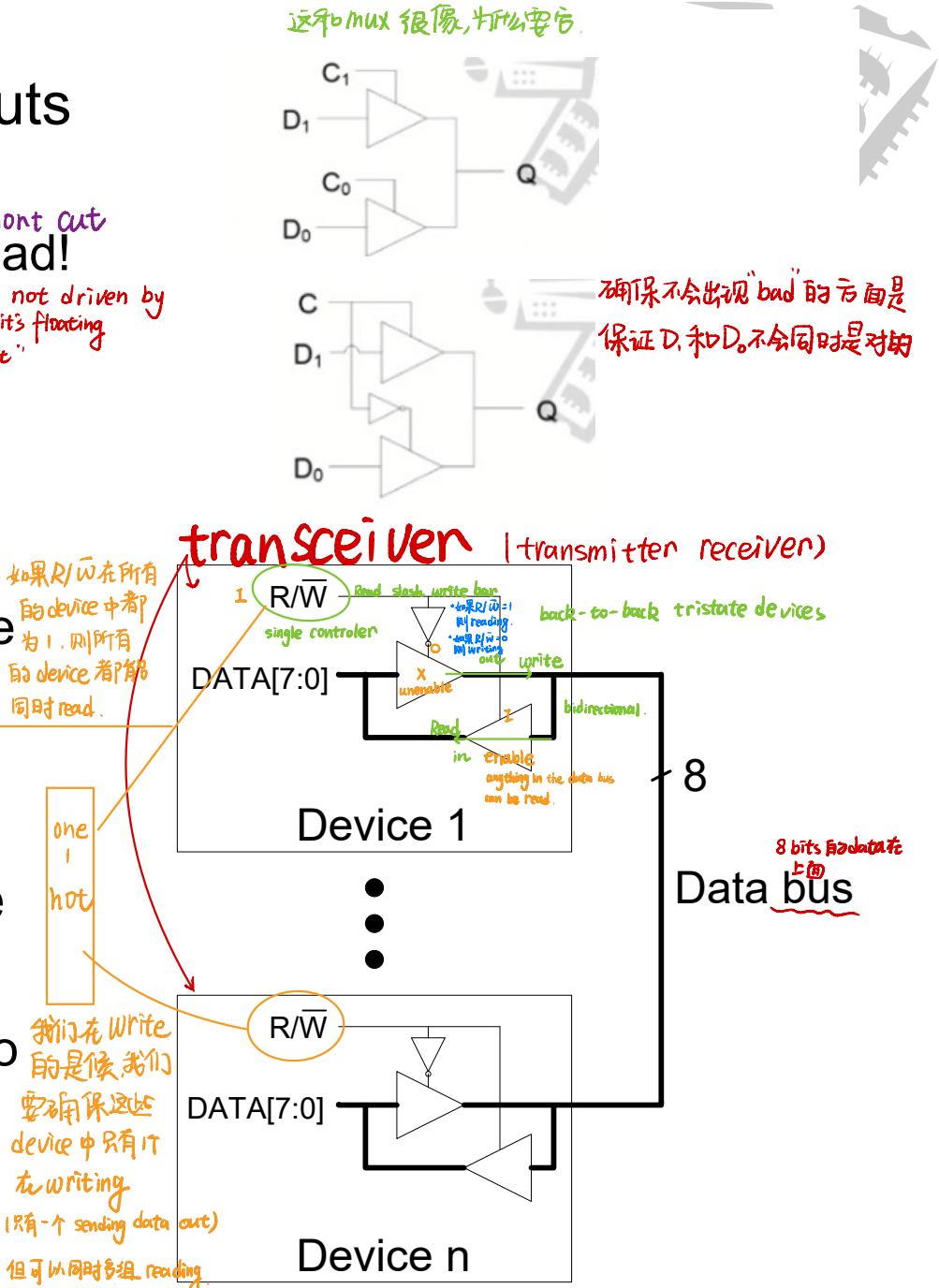
- $C_1 = C_0 = 1$ and $D_1 \neq D_0 \rightarrow \text{bad!}$
- $C_1 = C_0 = 0 \rightarrow Q = ?$ *the output is not driven by any of input, it's floating "indeterminant"*
- $D_1 = D_0 \rightarrow \text{OK!}$

- Typically, tri-state control signals are **one-hot**

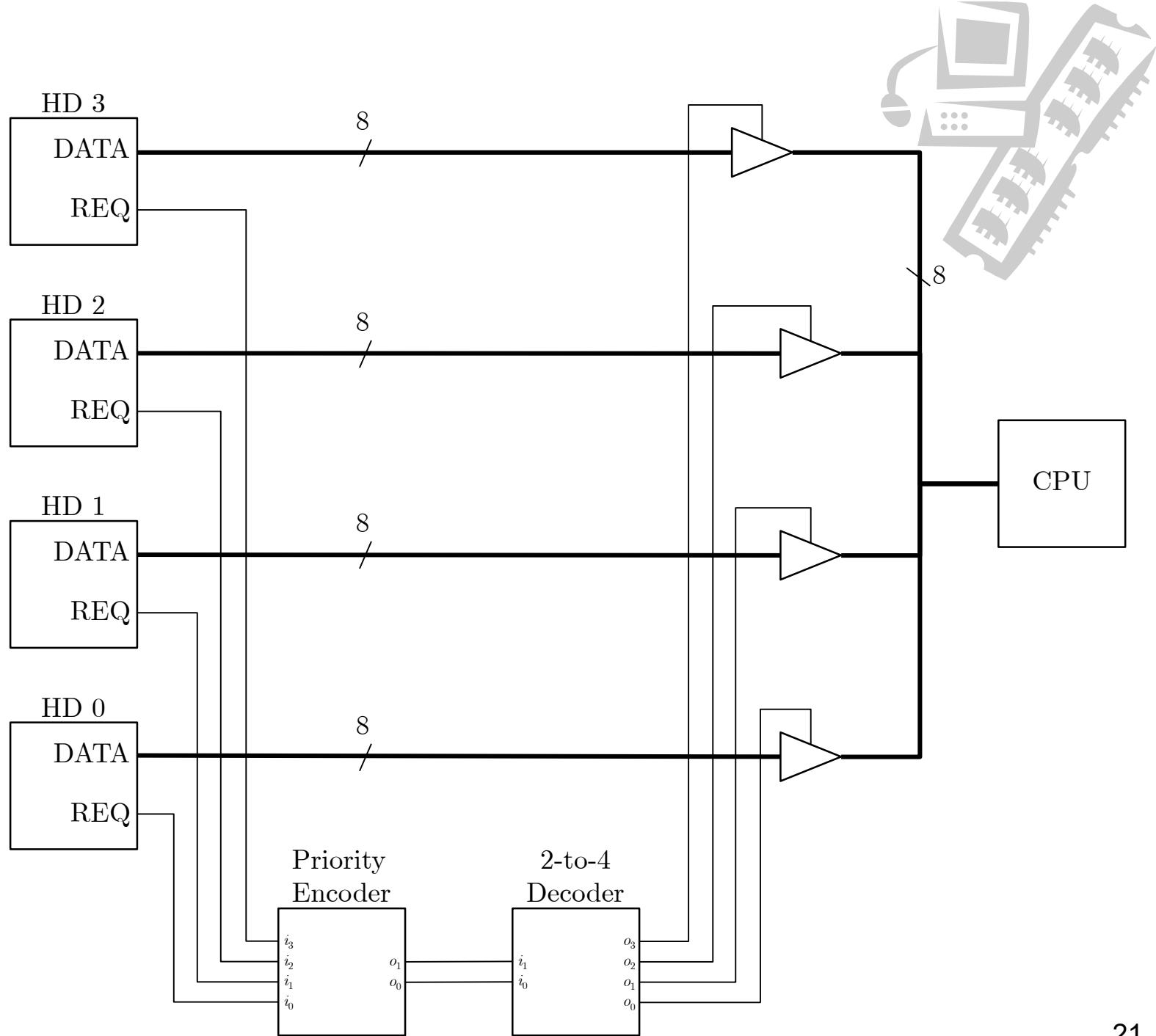
- Results in exactly one device connected to the output at a time

- Multiple devices can communicate over a single shared wire

- Only one device is allowed to drive the wire at a time
- A functional group of such wires is called a **bus**.



Tying it all together...



A B C D E

